CS2023 manuscript No.
(will be inserted by the editor)

Portrait Sketch Generation Based On Pix2pix Network

Zhao Kewei · Wei Yuhe · Lei Yuanwei

Abstract Portrait sketch generation aims to generate high-quality sketches from simple hand-drawn lines, which requires a generation network from tensors to tensors. In this project, we implement a traditional generative adversarial network model: $pix2pix$, and applies it to train and generate our sketch pieces. Then we add some optimization methods like $ConditionalTraining$ and $NoiseRemoval$ to make the generated sketches more realistic with higher scores.

Keywords Pix2pix · Sketch generation · Model optimization

1 Introduction

In this section, we will briefly introduce the work content of portrait sketch generation challenge and the GAN model.

1.1 Portrait sketch generation challenge

Portrait sketching is an art form that uses black, white, and gray to create an image of a person through the use of lines. Different artistic styles and techniques can convey various emotions and moods. However, traditional portrait sketching heavily relies on the personal drawing skills of the artist and requires significant time and effort for each drawing. Thanks to the development of image style transfer, AI-assisted portrait sketching can significantly reduce the cost of creating portraits. However, the most common method of generating portrait sketches is based on input photos, which may not be available in all situations.

Zhao Kewei
520021910674

Wei Yuhe
520021910313

Lei Yuanwei
520030910062

In this challenge, we will solve the problem of generating portrait sketches without photos, allowing ordinary people to quickly obtain high-quality portrait sketches that meet their own requirements.We will train on the provided dataset which includes both normal facial lines and professional portrait sketches, using rough facial line images as input to generate corresponding portrait sketches as output. The generated results are then to be uploaded to the competition platform and evaluated using two performance metrics.

1.2 GAN introduction

GAN is a deep learning model, also known as Generative Adversarial Networks. It consists of two neural networks: a generator network and a discriminator network.

The generator network generates new data by learning the distribution of training data. The discriminator network attempts to distinguish between the data generated by the generator and the actual training data. During the training process, the two networks confront each other, the generator network attempts to deceive the discriminator network, making it unable to accurately distinguish between the generated data and the actual training data, while the discriminator network attempts to correctly identify which data is real. The relationship between the two forms an adversarial network, hence it is called an adversarial network.

Through continuous iterative training, the generator network gradually learns how to generate more realistic data, while the discriminator network gradually becomes more accurate. Finally, the generator network can generate new data similar to the training data, which can be used in image generation, video generation, natural language processing and other fields.

GAN is a very powerful deep learning model with a wide range of applications, including image generation, video generation, speech synthesis, image style conversion, and so on. Meanwhile, the training of GAN is also very complex, requiring consideration of multiple factors, such as the quality of training data and the design of network structure.

2 Pix2pix network model

Pix2pix is a general framework for image translation based on conditional GAN. It realizes the generalization of model structure and loss function, and has achieved remarkable results on many image translation data sets. Its core technology has three points: conditional GAN based loss function, U-Net based generator and PatchGAN based discriminator.

There is a type of task called image to image translation, where the input and output are images from two different sets (set as A and B), and we generally believe that they have corresponding relationships. For example, input black and white photos (A) to output color photos (B), input outline photos (A) to output color filled photos (B), and so on. The pix2pix model handles this type of task.

Pix2pix is essentially a special type of conditional GAN. Pix2pix draws inspiration from the ideas of cGAN. When inputting into the G network, cGAN not only inputs noise, but also inputs a condition, and the fake images generated by the G network will be affected by specific conditions. So if an image is used as a condition, the generated fake images correspond to this condition images, thus achieving an image to image translation process. The principle of pixfix is shown in the figure.1

Generator G uses a U-Net structure, where the input contour map x is encoded and decoded into a real image. Discriminator D uses the conditional discriminator PatchGAN, which determines whether the generated image $G(x)$ is false and the real image is true under the conditions of contour map x.

Pix2pix is based on cGAN for image translation, as cGAN can guide image generation by adding conditional information. Therefore, in image translation, the input image can be used as a condition to learn the mapping from the input image to the output image, and obtain the specified output image. However, other GAN based image translation algorithms rely on other constraints to guide image generation, as the generator

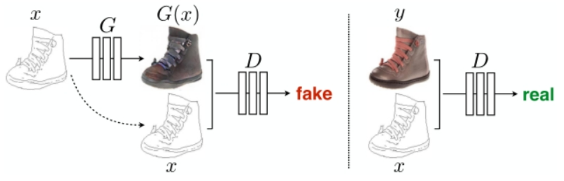

Fig. 1 The training process of pix2pix

of the GAN algorithm is based on a random noise that makes it difficult to control the output. This is the difference between pix2pix and other GAN based image translation methods.

3 Implements

In this section, implementation details will be introduced, including data pre-processing, network architectures, parameter settings and optimization methods. To get more information, please check our source code files *main.py* and *model.py*.

3.1 Pre-processing

Original input is RGB pictures, and each picture has the size $[3, 1024, 1024]$, which is too large to train. Therefore we resize it to size $[3, 256, 256]$. With this size, most critical information are preserved, and the overall expenses significantly decreases. Also we apply normalize function to make the data more convenience for training. All these operations can be achieved using $transforms.Compose([transforms.Resize([256, 256]), transforms.ToTensor(), transforms.Normalize()])$.

To pairwise training lines and sketches, we rewrite the class *torch.utils.data.Dataset* to adjust specific data provided for generation.

3.2 Generator

Generator is the key part of the whole architecture in sketch generation. We apply a U-net architecture to our generator, which is introduced in [2]. The net consists of 6 down-sampling layers, 5 up-sampling layers and a last transpose convolution layer. Each down-sampling layer performs a 2D convolution with $stride = 2$ to reduce input size by half, and then use *LeakyRelu* and *BatchNorm* functions to further process data. To the contrast, each up-sampling layer performs a transpose convolution, doubling the input size. The generator accepts an input tensor with size $[3, 256, 256]$, and will output a tensor with the same size.

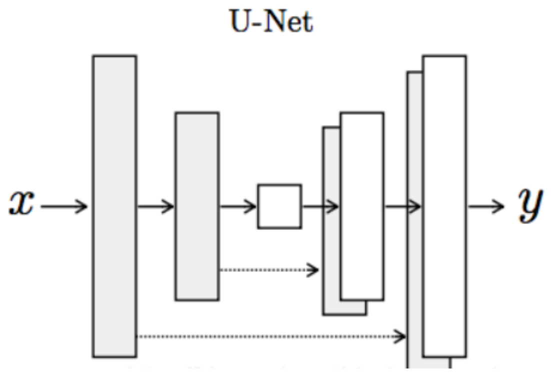

Fig. 2 U-net structure

In general U-nets (Figure 2), up-sampling layer will take two parts as input: one is the output of the previous layers, and the other is the intermediate output from corresponding down-sampling layer. We also take this design in our network. Thus extra information is required during forward propagation. To deal with this, we need to carefully backward parameters, which is introduced in section 3.4

3.3 Discriminator

Discriminator is simpler than generator. It only consists of 2 down-sampling layers and extra 2 convolution layers with $stride = 1$. Under this structure, discriminator can better grab the features of data by convolution.

According to the theory of $PatchGAN$, our discriminator will output a $1*60*60$ tensor based on $6*256*256$ input. The input is a pairwise of line and sketch, each has the size $[3, 256, 256]$.The sketch is from either realistic sketch or generated sketch, and the discriminator will judge whether sketch is real with line.

The brief structure hierarchy is shown in Figure 3

3.4 Loss function

We take the loss function introduced in [1]. The objective function is:

$$G^* = argmin_G max_D [L_{cGAN}(G, D) + \lambda L_{L1}(G)], ...(1)$$

where $L_{cGAN}(G, D)$ is the loss function (BSE loss) in cGAN and $\lambda L_{L1}(G)$ is the L1 loss for generator G.

This objective function conveys the link between generator G and discriminator D: G tries to generate more realistic sketches to cheat D, while D tries to distinguish fake photos generated by G from real photos.

The details of loss calculation is shown as below:

$real_D = D(lines, sketches)$

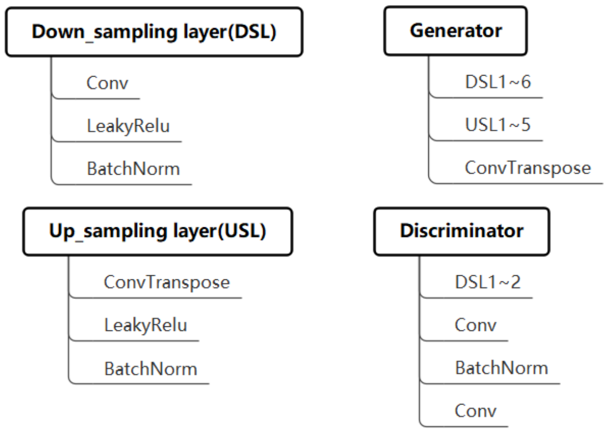

Fig. 3 Our network architecture

$realLoss_D = loss_{bce}(real_D, ones\_like(real_D))$
$realLoss_D.backward()$
$fake_D = D(lines, output_G.detach())$
$fakeLoss_D = loss_bce(fake_D, zeros\_like(fake_D))$
$fakeLoss_D.backward()$
$loss_D = realLoss_D + fakeLoss_D$
$BCELoss_G = loss_{bce}(output_D, ones\_like(output_D))$
$L1Loss_G = loss_{L1}(output_G, sketches)$
$loss_G = 0.1 * (BCELoss_G + \lambda * L1Loss_G)$
$loss_G.backward()$

Here, for discriminator D, $realLoss_D$ calculates the distance from D's prediction to '1' ($1*60*60$ tensor with all elements set to 1) in real sketches, and $fakeLoss_D$ calculates the distance from D's prediction to '0' in generated sketches.

For generator G, $BCELoss_G$ evaluates the performance of G in cheating D, and $L1Loss_G$ evaluates the similarity between generated sketches and real sketches, making sure that generated sketches will not only cheat D but also be more realistic.

As up-sampling layers takes more intermediate information(including values and gradients), it's unsuitable to backward all parameters at the same time, which will cost too much space. Separated backward is applied to reduce space complexity, as well as increase robust of the whole network.

3.5 Optimization methods

In this section, we will introduce two optimization methods applied to our network. And in section 4, experimental results with/without optimization will be shown.

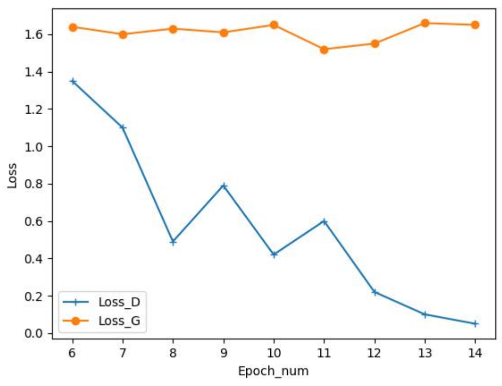

Fig. 4 Loss change for discriminator D and generator G during a period of training

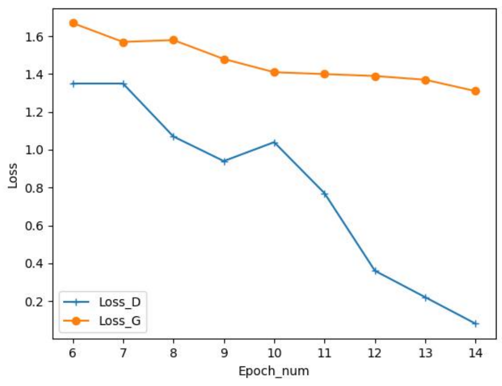

Fig. 5 Loss change for discriminator D and generator G during a period of training with conditional learning

3.5.1 Conditional training

Due to different structure and task difficulty, the strength of generator and discriminator can be different. Generator with 12 layers are required to do a full-size reflection from $[3, 256, 256]$ to $[3, 256, 256]$, while discriminator with 5 layers only needs to reflect tensors from $[3, 256, 256]$ to $[1, 60, 60]$, thus discriminator is easier to reach a stable, convergent loss.

As generator is "weaker" than discriminator, there may be abnormal fluctuations in $Loss_G$ during training after training for a few epochs. Traditional GAN training [3] applies an interleaving training method, which trains discriminator and generator alternately, step by step. However, in sketch generation, unsuitable parameters can probably cause the reverse of gradients of not only generator but discriminator. According to Fig 4, where the strategy is interleaving training in traditional GAN. Loss_D is decreasing generally while Loss_G maintains.

Above all, we introduce a new strategy for training named condition training. We train generator for every epoch, but train discriminator under specific condition:

- *Fixed timer*: Train discriminator or generator one time for every N epochs.
- *Condition judgment*: If $Loss_D$ is very small while $Loss_G$ is getting even larger, the discriminator will be locked and only generator will be trained. To the contrast, generator can be also locked if it seems to be much stronger than discriminator.

By conditional learning, the problem of no-decrease loss is improved, which is shown in Fig 5. And due to less update of parameter, time and space expenses of our network is also reduced. Detailed parameter setting and corresponding result is shown in section 4.1,

3.5.2 Noise Removal

In GAN, generator tends to learn a deterministic reflection from input to output, thus may cause an overfitting in test samples, leading to a lot of noise on generated sketch.

To deal with noise, we try a series of methods, including:

- *Random dropout*: We add dropout layers into upsampling layer. During training, some nodes will be randomly shut down, increasing randomness to avoid overfitting.
- *Grayscale sketches*: Generated sketches has colorful noise. We change generated sketches(RGB style) to gray style, and then remove noise by comparing noisy sketches with normal sketches.

By noise removal, the quality of generated sketches is further increased. After optimization, except for some particular features like eyes, other features are well generated by our network.

4 Experiments

To test the performance of our pix2pix network, we compare it to normal convolution network, and then change hyper-parameters to get multiple sets of generated sketches. After this, we load $ConditionalTraining$ and $NoiseRemoval$ methods to the network.

In this section, we mainly evaluate the quality of generated sketch toward 2 ranking methods:

1) FID: The performance of GAN can be discribed as the similarity of Gaussian distributions between generated sketches with real sketches, which is evaluated by FID. Lower value of FID means better performance.

– $SSIM$: $SSIM$ is used to measure the similarity between two sketches. $SSIM$ is a perception model, which is closer to the intuitive feelings of human eyes. Larger value of $SSIM$ means better performance.

4.1 Pix2pix performance

Figure 6 shows different generated sketches between different networks after training for 50 epochs. A normal 5 layer convolution network loses most features in training sets and only learns an average information of all sketches. No matter what line photo as input, convolution network will give almost the same output: an average face. However, pix2pix network indeed captures the common features for different sketches and can generate an approximate sketch from a new line.

4.2 Hyper parameter selection

Hyper parameters can significantly influence the overall performance of a network. In our pix2pix network, we set a bench of hyper parameters including:

– $BATCHSIZE$: The batches per train. Appropriate batch_size may speedup training process with better result. But in this project, it seems to have no influence to generation.
– $LEARNING\_RATE$: Learning rate parameter lr for Adam optimizer. Different lr for discriminatorand generator may cause problem mentioned in section 3.5.
– $LAMBDA$: Coefficient for generator's L1 loss. When λ gets larger, the generator considers more to generate sketches closer to which in training set rather than cheat the discriminator.
– $TIMER$: Timer for conditional training. Discriminator will be trained once for every $TIMER$ epochs.
– $LOSS\_THRESHOLD$: Threshold for conditional training. Discriminator will be trained only when its loss is larger than this threshold.

Above all these parameters, $LAMBDA$ is the critical influence factor. Table 1 shows the average FID scores drops slightly as λ increases.

LAMBDA	Avg_FID
8	1.97
20	1.89
100	1.70

Table 1 FID scores under different λ

4.3 Optimization methods

Conditional training improves the loss gained during training. $Loss_G$ decreases for about 30% - 40% after appending conditional training. After conditional training, sketch definition increases a lot, but it also generates more noise in sketches.

Noise removal strengthens sketch quality during final generation process. It changes colorful noise to gray, and remove shadows in the edge of sketches.

Figure 7 shows the performance for our network with/without optimization methods. Here, CT stands for $ConditionalTraining$ and NR stands for $NoiseRemoval$. In general, generated sketches with both methods have smallest FID and highest SSIM.

Method	Avg_FID	Avg_SSIM
pix2pix	1.70	0.64
pix2pix+CT	1.45	0.66
pix2pix+CT+NR	1.34	0.66

Table 2 FID scores under different optimization methods

5 Conclusion

In this paper, we introduce theories of sketch generation and useful deep network pix2pix for generation tasks. To finish the task, we build a full pix2pix architecture with generator and discriminator. To further optimize the quality of generated sketches, we implements conditional training as well as noise removal method to our network, and finally we get better results.

However, pix2pix network is still too weak to deal with some details, like hairs and eyes, which makes sketches unperfect. Newer networks, like styleGAN, may have better results than pix2pix.

After all, we learn a lot through this challenging sketch generation task. We thank professor and TA, as well as all people helping us in this lecture.

References

1. Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A.Efros. Image-to-Image Translation with Conditional Adversarial Networks. https://arxiv.org/abs/1611.07004 (2016)
2. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. https://arxiv.org/abs/1505.04597
3. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. https://arxiv.org/abs/1406.2661

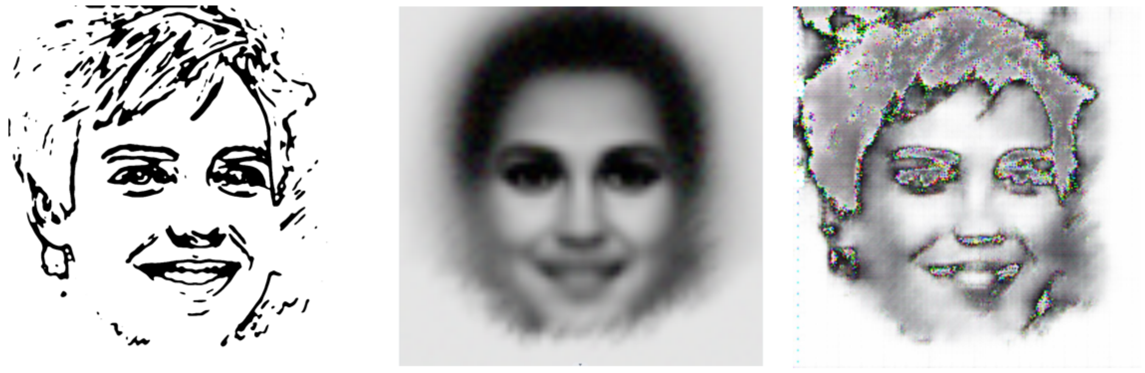

Fig. 6 Performance comparison between normal convolution network and basic pix2pix network. The left picture is a line for sketch generation, the middle pictute is the sketch generated by convolution network and the right picture is the sketch generated by pix2pix network.

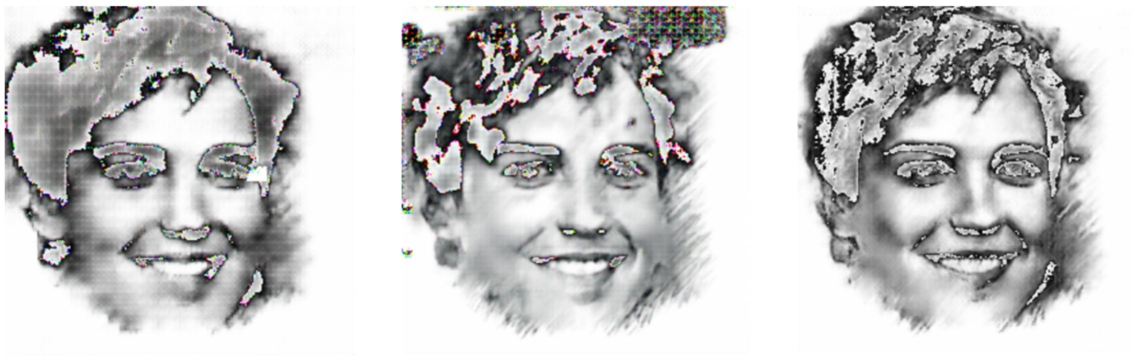

Fig. 7 Generated sketches among basic pix2pix network(the left one), pix2pix + conditional training(the middle one) and pix2pix + conditional training + noise removal(the right one). LAMBDA is fixed to 100, and EPOCHS is set to 200